สุทธิศักดิ์ พงศ์ธนาพาณิช

เปิดโลกการเข้ารหัสข้อมูลด้วย ภาษาซี

ตอน Huffman coding (ตอนจบ)

ตอนสุดท้ายของกรรมวิธีการเขียนโปรแกรมบีบอัดข้อมูลด้วย C

จากตอนที่ผ่านมาผมได้อธิบายถึงอัลกอริทึมของการบีบอัดข้อมูลแบบ Huffman coding และในตอนท้ายของบทความได้เสนอแนะขั้นตอนการออกแบบรูทีนสำหรับการบีบอัดข้อมูล (CompressFile) และรูทีนการขยายข้อมูล (ExpandFile) ดังนั้นในบทความตอนนี้ผมจึงขอจบการบีบอัดข้อมูลด้วยวิธี Huffman coding ด้วยการเสนอโปรแกรม HUFFMAN.C พร้อมทั้งอธิบายการทำงานของแต่ละรูทีนที่เกี่ยวข้องเพื่อเป็นแนวทางให้ผู้อ่านสามารถติดตามหรือปรับปรุงความสามารถของโปรแกรมให้มากยิ่งขึ้นก็ได้

ในการออกแบบโปรแกรมผมได้แยกจัดเก็บรูทีนต่างๆ ลงในหลายๆ โปรแกรมโดยแบ่งแยกตามกลุ่มของการทำงาน เนื่องจากผมได้แยกไฟล์ BITIO.C ออกจากไฟล์อื่นๆ เพราะรูทีนต่างๆ ในไฟล์นี้สามารถที่จะใช้ร่วมกันได้กับการบีบอัดข้อมูลทุกวิธี ดังนั้นขอให้ผู้อ่านติดตามรายละเอียดของไฟล์นี้ได้จากวารสารไมโครฯ ในตอน Shannon-Fano coding ซึ่งในการสร้างโปรแกรมบีบอัดข้อมูลด้วยวิธี Huffman coding ในตัวอย่างต่อไปนี้ประกอบด้วยไฟล์ต่างๆดังนี้

ไฟล์ MAIN.C

เป็นส่วนของโปรแกรมไดรเวอร์ที่ผมใช้ในการทดสอบการบีบอัดข้อมูลด้วยวิธี Huffman coding ซึ่งประกอบด้วยไฟล์เฮดเดอร์ MAIN.H และรูทีนต่างๆดังนี้

main()

เป็นรูทีนหลักที่รับคำสั่งจากผู้ใช้และทำการเรียกรูทีนบีบอัดหรือขยายข้อมูลตามค่าของพารามิเตอร์ที่กำหนดให้กับโปรแกรม

usage_exit()

กรณีที่ผู้ใช้เรียกใช้โปรแกรมไม่ถูกต้อง เช่นกำหนดค่าพารามิเตอร์ผิดหรือไม่ครบตามที่กำหนด เราก็จะแนะนำวิธีการเรียกใช้โปรแกรมแก่ผู้ใช้โดยผ่านทางรูทีนนี้

print_ratios()

ทำหน้าที่รายงานผลอัตราส่วนการบีบอัดข้อมูลที่ได้ โดยใช้สูตรดังนี้

```
ratio = 100  (int) ( output_size *
100L/     input_size );
```

long file_size();

ทำหน้าที่รายงานขนาดของไฟล์ที่กำหนดในหน่วยไบต์

ไฟล์ HUFFMAN.C

รูทีนต่าง ๆ ที่นอกเหนือจากที่ทำหน้าที่อ่านหรือเขียนข้อมูลทีละบิตหรือ

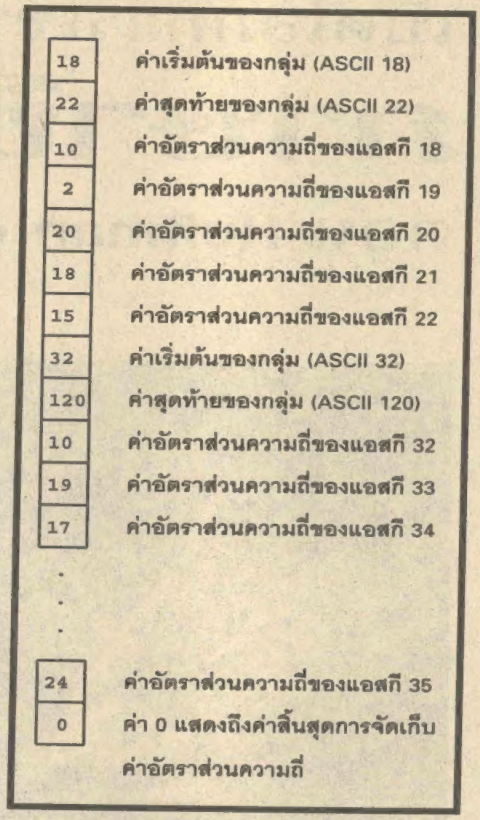
หลายๆ บิตลงในไฟล์ซึ่งผมได้รวมกลุ่มเอาไว้ในไฟล์ BITIO.C แล้วนั้น สำหรับรูทีนต่างๆ ที่ทำหน้าที่ตามขั้นตอนต่างๆ สำหรับการบีบอัดข้อมูลด้วยวิธี Huffman coding ก็จะถูกจัดเก็บรวมเอาไว้ในไฟล์นี้ ดังนั้นถ้าหากผู้อ่านต้องการที่จะนำรูทีนที่ทำหน้าที่บีบอัดข้อมูลด้วยวิธี Huffman coding ไปใช้กับโปรแกรมอื่นๆ ก็ให้ทำการรวมเอารูทีนทั้งหมดในไฟล์นี้และไฟล์ BITIO.C เข้าไปไว้ด้วยกัน สำหรับรูทีนต่างๆ มีหน้าที่และการทำงานดังนี้

count_bytes()

รูทีนนี้ทำหน้าที่ตามขั้นที่ 2 ของการบีบอัดข้อมูล คือทำการนับค่าความถี่ของตัวอักษรทุกๆตัวในไฟล์โดยนับตั้งแต่เริ่มต้นจนกระทั่งสิ้นสุดไฟล์ และจัดเก็บค่าความถี่ของแต่ละตัวอักษรลงในตัวแปร counts[ ]

scale_counts()

หลังจากที่นับค่าความถี่ของตัวอักษรแล้วเราก็จะทำการแปลงค่าความถี่ที่ได้ให้อยู่ในรูปแบบอัตราส่วนของค่าความถี่สูงสุด เพื่อให้สามารถจัดเก็บค่าอัตราส่วนความถี่นี้ลงในตัวแปรขนาด unsigned integer ได้ ซึ่งข้อดีของการแปลงค่าอัตราส่วนความถี่มีหลายประการ ดังนี้ ประการแรกเราสามารถลดการคำณวนทางคณิตศาสตร์ลงได้ในขณะที่สร้างแผนภูมิต้นไม้ซึ่งจะทำให้โปรแกรมสามารถทำงานได้เร็วขึ้น ประการสุดท้ายเราสามารถลดพื้นที่ที่ใช้จัดเก็บแต่ละโหนดลงได้ สำหรับการทำงานของรูทีนก็เริ่มด้วยการหาค่าความถี่สูงสุด หลังจากนั้นจึงทำให้เป็นค่าอัตราส่วนของ 255 โดยการหารค่าความถี่สูงสุดด้วย 255 แล้วบวกอีกหนึ่งเพื่อไม่ให้ค่าเป็น 0 และจัดเก็บค่าที่ได้ลงในตัวแปร max_count แล้วจึงทำการแปลงค่าความถี่ของทุกๆ ตัวอักษรโดยการหารค่าความถี่ด้วย max_count และ



รูปที่ 1 แสดงการจัดเก็บค่าอัตราส่วนความถี่ลงในไฟล์บีบอัดข้อมูล

จัดเก็บค่าที่ได้ลงในสมาชิกของตัวแปรโครงสร้าง nodes[ ].count

output_counts()

ส่วนในขั้นตอนที่ 4 ของการบีบอัดข้อมูลก็โดยการจัดเก็บค่าอัตราส่วนความถี่ลงในไฟล์บีบอัดทั้งนี้เพื่อให้โปรแกรมขยายขนาดข้อมูลสามารถนำข้อมูลตัวนี้มาใช้สร้างแผนภูมิต้นไม้ได้อีกครั้ง เพื่อให้สามารถทำการถอดรหัสข้อมูลได้อย่างถูกต้อง สำหรับการจัดเก็บผมจะไม่ใช้วิธีจัดเก็บค่าทั้ง 256 ตัวอักษรเพราะโดยทั่วไปไฟล์จะประกอบด้วยตัวอักษรเพียงบางกลุ่ม เช่นในไฟล์ประเภทข้อความ (text file) ก็จะประกอบด้วยตัวอักษรตั้งแต่รหัสแอสกี 32 ถึง 126 เป็นต้น ดังนั้นเพื่อเป็นการประหยัดพื้นที่จึงออกแบบรูปแบบการจัดเก็บเสียใหม่แบบเป็นกลุ่มๆและแต่ละกลุ่มจะนำด้วยค่าแอสกีเริ่มต้นและแอสกีสุดท้ายของกลุ่ม ดังในรูปที่ 1

build_tree()

เมื่อเสร็จจากขั้นตอนที่ 4 เราก็ทำการสร้างแผนภูมิต้นไม้ Huffman ด้วยรูทีนนี้ ซึ่งการทำงานของรูทีนนี้เริ่มต้นด้วยการกำหนดค่าของโหนดที่ 513 ให้มีค่าเท่ากับ 65,535 ซึ่งโหนดปกติจะมีค่าที่ไม่เกินกว่าค่านี้แน่นอน และกำหนดให้โหนดที่น้อยที่สุดเป็น 513 แล้วจึงเริ่มต้นค้นหา 2 โหนดที่มีค่าน้อยที่สุดเพื่อสร้างโหนดแม่ต่อไปเรื่อยๆ ซึ่งโหนดใดที่มีค่ามากกว่า 0 เราจะเรียกว่าโหนดแอคทีฟ (activenode) ในการสร้างแผนภูมิต้นไม้ก็จะมีการวนซ้ำเพื่อหา 2 โหนดที่มีค่าน้อยที่สุดเรื่อยๆ ดังนั้นถ้าหากโปรแกรมพบโหนดแอคทีฟเหลือเพียงโหนดเดียวก็จะสิ้นสุดการวนซ้ำและแสดงว่าแผนภูมิต้นไม้ได้ถูกสร้างเสร็จสมบูรณ์แล้ว แต่ถ้าหากยังคงพบโหนดแอคทีฟ 2 โหนดอีกโหนดใหม่ก็จะถูกสร้างขึ้นมาโดยมีค่าความถี่เท่ากับผลรวมของโหนดลูก 2 โหนด ซึ่งโหนดใหม่ที่ถูกสร้างขึ้นมานี้ก็จะถูกกำหนดหมายเลขโหนดด้วยค่าของ next_free และโหนดแอคทีฟ 2 และโหนดแอคทีฟ 2 โหนดที่ถูกพบก็จะถูกกำหนดสถานะเป็น inactive โดยการกำหนดให้มีค่าน้ำหนักเป็น 0 เพราะโหนดที่เป็น 0 จะถูกพิจารณาเป็นโหนดที่จะไม่ถูกเลือกอีกต่อไป

convert_tree_to_code()

หลังจากที่สร้างแผนภูมิต้นไม้ Huffman เสร็จแล้วเราก็จะทำการถอดรหัสตัวอักษรแต่ละตัวจากแผนภูมิต้นไม้ ซึ่งในการถอดรหัสเราจะเริ่มจากตำแหน่ง ROOT ของแผนภูมิและสิ้นสุดที่ทุกกิ่งก้านล่างสุดของแผนภูมต้นไม้ ซึ่งจะเห็นว่างานลักษณะอย่างนี้เป็นงานที่มีการทำอย่างซ้ำซ้อนโดยเริ่มต้นจากกิ่งก้านซ้ายมือและต่อมา

ทางด้านกิ่งก้านขวามือไปเรื่อยๆ จนหมดทุกๆตัวอักษรที่มีในแผนภูมิต้นไม้ ดังนั้นวิธีที่ง่ายที่สุดสำหรับงานประเภทนี้ก็คือการใช้ฟังก์ชันแบบเรียกตัวเอง (recursive function) และกำหนดให้เรียกตัวเองโดยมีทั้งโหนดซ้าย (nodes[].child_0) และโหนดขวา (nodes[].child_1)

compress_data()

ขั้นตอนสุดท้ายของการบีบอัดข้อมูล ก็คือการบีบอัดข้อมูลของแต่ละตัวอักษรโดยใช้การเข้ารหัสด้วยโค้ดที่มีความยาวไม่คงที่ ซึ่งเป็นแนวคิดพื้นฐานของการบีบอัดข้อมูลโดยทั่วไป ซึ่งรูทีนนี้ก็ทำการอ่านตัวอักษรแต่ละตัวจากไฟล์แล้วทำการเข้ารหัสตัวอักษรด้วยโค้ดที่ได้จากขั้นตอนที่ผ่านมา หลังจากนั้นก็จัดเก็บโค้ดที่ได้ลงในไฟล์บีบอัดซึ่งในการจัดเก็บก็ยังคงใช้รหัสแอสกีเหมือนเดิม

input_counts()

ส่วนในขั้นตอนการขยายข้อมูลเราก็ต้องมาเริ่มต้นที่ขั้นตอนการอ่านค่าอัตราส่วนความถี่ที่ถูกจัดเก็บลงในไฟล์ (ในส่วน เฮดเดอร์) เพื่อที่จะนำค่าดังกล่าวมาสร้างแผนภูมิต้นไม้อีกครั้งด้วยรูทีน build_tree()หลังจากนั้นก็ย้อนกลับขบวนการเดิมคือการถอดรหัสแต่ละตัวอักษรจากแผนภูมิต้นไม้เพื่อที่จะแปลงโค้ดที่มีความยาวไม่คงที่ให้กลับเป็นโค้ดตัวเดิมก่อนการบีบอัดด้วยรูทีน convert_tree_to_code()

expand_data()

รูทีนนี้ทำหน้าที่ขยายข้อมูลที่ถูกบีบอัด การทำงานของรูทีนนี้ค่อนข้างยากกว่าการบีบอัดข้อมูล โดยเริ่มจากการอ่านข้อมูลจากไฟล์บีบอัดทีละบิตแล้วจึงเดินตามแผนภูมิต้นไม้ทีละกิ่งโดยถ้าหากบิตที่อ่านได้มีค่าเป็น 1 ก็เดินตาม nodes[ ].child_1 และในทางกลับกันถ้าหากบิตดังกล่าวมีค่าเป็น 0 ก็จะเดินตามกิ่ง nodes[ ]. child_1 เมื่อเดินถึงกิ่งล่างสุดก็แสดงว่าโค้ดที่เราอ่านขึ้นมาทั้งหมดตรงกันกับตัวอักษรในโหนดนั้นๆ

การทำงานของโปรแกรม

เมื่อผู้อ่านรันโปรแกรมในเอนไวรอนเมนต์ของ Quick C หรือดอส ถ้าหากกำหนด option เป็น -c โปรแกรมก็จะทำการบีบอัดไฟล์ infile และจัดเก็บไฟล์ที่ถูกบีบอัดลงในไฟล์ outfile ซึ่งขั้นตอนการบีบอัดข้อมูลด้วยวิธี Huffman coding นั้นได้ถูกรวมเอาไว้ในรูทีน CompressFile ซึ่งจะเริ่มต้นโดยการจองพื้นที่หน่วยความจำให้กับตัวแปร counts, nodes และ codes เพื่อใช้ในการทำงานของโปรแกรมจากนั้นก็เริ่มต้นขบวนการบีบอัดข้อมูลโดยการนับค่าความถี่ของแต่ละตัวอักษรโดยรูทีน count_bytes จากนั้นค่าความถี่ของตัวอักษรทุกตัวที่มีในไฟล์ก็จะถูกจัดเก็บลงในตัวแปร counts จากนั้นก็ทำการแปลงค่าความถี่ให้อยู่ในรูปแบบค่าอัตราส่วนเพื่อช่วยลดเวลาการคำณวนและลดพื้นที่การจัดเก็บโดยรูทีน scale_counts ซึ่งผลลัพธ์ที่ได้จะถูกจัดเก็บลงในตัวแปร nodes จากนั้นก็ทำการจัดเก็บค่าอัตราส่วนความถี่ลงในไฟล์โดยรูทีน output_counts ซึ่งค่าที่จัดเก็บนี้เหมือนกับเป็นส่วน header ของไฟล์เพราะค่านี้จะถูกนำกลับมาใช้ใหม่ในขั้นตอนการขยายขนาดไฟล์สู่ปกติ จาก

```
void CompressFile( input, output )
FILE *input;
BIT_FILE *output;
{
    unsigned long *counts;
    NODE *nodes;
    CODE *codes;
    int root_node;

    counts = (unsigned long *) calloc( 256, sizeof( unsigned long ) );
    if ( counts == NULL )
        fatal_error( "Error allocating counts array\n" );
    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
            fatal_error( "Error allocating nodes array\n" );
    if ( ( codes = (CODE *) calloc( 257, sizeof( CODE ) ) ) == NULL )
            fatal_error( "Error allocating codes array\n" );
    count_bytes( input, counts );
    scale_counts( counts, nodes );
    output_counts( output, nodes );
    root_node = build_tree( nodes );
    convert_tree_to_code( nodes, codes, 0, 0, root_node );
    compress_data( input, output, codes );
    free( (char *) counts );
    free( (char *) nodes );
    free( (char *) codes );
}
```

รูปที่ 2

```
void ExpandFile( input, output )
BIT_FILE *input;
FILE *output;
{
    NODE *nodes;
    int root_node;

    if ( ( nodes = (NODE *) calloc( 514, sizeof( NODE ) ) ) == NULL )
        fatal_error( "Error allocating nodes array\n" );
    input_counts( input, nodes );
    root_node = build_tree( nodes );
    expand_data( input, output, nodes, root_node );
    free( (char *) nodes );
}
```

รูปที่ 3

นั้นก็เป็นขั้นตอนการสร้างแผนภูมิต้นไม้ huffman จากค่าความถี่ของตัวอักษรที่หาได้โดยรูทีน build_tree ซึ่งแผนภูมิต้นไม้นี้จะถูกนำไปแปลงให้เป็นโค้ดโดยรูทีน convert_tree_to_code ซึ่งค่าตามตำแหน่งโหนดต่างๆ ที่ถูกจัดเก็บในตัวแปร nodes ก็จะถูกแปลงให้เป็นโค้ดไบนารี่ที่มีความยาวแปรเปลี่ยนได้และจัดเก็บในตัวแปร codes สุดท้ายก็จะเป็นวิธีการเข้ารหัสตัวอักษรโดยวิธี huffman coding โดยโค้ดที่ได้จะถูกนำมาแทนที่รหัสแอสกีโดยรูทีน compress_data ซึ่งโค้ดใหม่ที่ได้ก็จะถูกจัดเก็บลงในไฟล์ outfile โดยใช้รหัสแอสกีเช่นเดิม โดยการเรียงบิตของโค้ดที่เข้ารหัสได้จนครบ 8 บิต แล้วแทนที่ด้วยรหัสแอสกีตามค่าที่ได้แล้วจึงค่อยจัดเก็บลงในไฟล์ เมื่อสิ้นสุดขบวนการบีบอัดข้อมูลเราก็ต้องยกเลิกหน่วยความจำที่จองเอาไว้ในตอนต้นก่อนจบรูทีน ในรูปที่ 2

แต่ถ้าหากผู้อ่านกำหนดค่า option เป็น -e ก็จะเป็นการสั่งให้โปรแกรมทำการขยายข้อมูลในไฟล์ infile และจัดเก็บผลลัพธ์ที่ได้ลงในไฟล์ outfile ซึ่งขั้นตอนการขยายขนาดข้อมูลกลับด้วยวิธี huffman coding ก็ได้ถูกรวมเอาไว้ในรูทีน ExpandFile ทั้งหมดซึ่งการทำงานก็เริ่มต้นด้วยการจองหน่วยความจำให้กับตัวแปร nodes จากนั้นก็เริ่มต้นการขยายขนาดข้อมูลโดยเริ่มต้นจากการอ่านข้อมูลในส่วน header ซึ่งเป็นค่าอัตราส่วนความถี่ขึ้นมาเพื่อนำมาใช้ในการสร้างแผนภูมิต้นไม้ huffman อีกครั้งโดยรูทีน input_counts จากนั้นก็ทำการสร้างแผนภูมิต้นไม้ huffman ด้วยรูทีน build_tree หลังจากนั้นก็มาถึงขั้นตอนสุดท้ายคือการแปลงโค้ดที่ถูกเข้ารหัสกลับคืนสู่สภาพเดิมโดยรูทีน expand_data โดยการอ่านทีละบิตและเดินตามกิ่งก้านของแผนภูมิต้นไม้ หลังจากนั้นก็จัดเก็บโค้ดที่แปลรหัสกลับคืนลงในไฟล์ outfile และก่อนจบการทำงานของรูทีนก็ต้องยกเลิกหน่วยความจำที่จองให้กับตัวแปร nodes เสียก่อน ซึ่งผู้อ่านจะพบว่าขั้นตอนการขยายขนาดข้อมูลจะง่ายกว่าขั้นตอนการบีบอัดข้อมูลอยู่บ้าง ดังแสดงลิสติงโปรแกรมในรูปที่ 3

```
QuickC :        qcl /W3 /Za /Fehuffman main.c bitio.c huff.c
Microsoft C :   cl /W3 /Za /Fehuffman main.c bitio.c huff.c
```

รูปที่ 4 แสดงคำสั่งในการสร้างไฟล์ .EXE

การใช้งาน

ผมได้ทดลองคอมไพล์โปรแกรมนี้ด้วย Microsoft QuickC 2.0 และ Microsoft C 6.0 โดยในที่นี้ผมตั้งชื่อโปรแกรมว่า HUFFMAN ดังนั้นคำสั่งในการสร้างไฟล์ .EXE สำหรับคอมไพล์เลอร์ทั้งสองจึงเป็นดังรูปที่ 4

ในการเรียกใช้ไฟล์ที่คอมไพล์เป็น .EXE แล้วในดอสก็สามารถกำหนดค่าของ command line ดังมีรูปแบบดังนี้

```
huffman infile outfile option
```

โดยที่ infile หมายถึงชื่อไฟล์ที่ต้องการบีบอัด

outfile หมายถึงชื่อไฟล์ที่เก็บข้อมูลที่ถูกบีบอัดแล้ว

option หมายถึงค่าพารามิเตอร์ที่จะกำหนดให้โปรแกรมทำการบีบอัดหรือขยายข้อมูล ดังนี้

-c กำหนดให้โปรแกรมทำการบีบอัดไฟล์

-e กำหนดให้โปรแกรมทำการขยายไฟล์

สุดท้ายก่อนที่จะจบเรื่อง Huffman coding ผมก็ขอเสนอผลของการบีบอัดข้อมูลไฟล์โปรแกรมตัวอย่างข้างล่างด้วยโปรแกรมตัวอย่างนี้ และในฉบับต่อไปก็ขอเสนอเทคนิคการบีบอัดข้อมูลที่พัฒนาจากวิธี Huffman coding นั่นก็คือวิธี Adpative Huffman coding ขอให้ติดตามกันต่อไปนะครับ (รูปที่ 5)

ซอฟต์แวร์

เปิดโลกการเข้ารหัสข้อมูลด้วยภาษาซี

ตอน Huffman coding (ตอนจบ)

โปรแกรม	ขนาดไฟล์ต้นฉบับ (ไบต์)	ขนาดไฟล์บีบอัด (ไบต์)	อัตราส่วนการบีบอัด (%)
MAIN.C	3,448	2,143	38%
BITIO.C	4,026	2,567	37%
HUFF.C	8,021	5,021	38%

รูปที่ 5

ลิสติง 1 MAIN.H

```
/*
*MAIN.H
*/
#ifndef_MAIN_H
#define_MAIN_H

voidCompressFile();
voidExpandFile();

externchar*Usage;
externchar*CompressionName;

#endif/*_MAIN_H*/
```

ลิสติง 2 MAIN.C

```
/*
*MAIN.C  Afterthisisbuiltintoacompressionprogramofanysort,the
*        programcanbecalledlikethis:
*
*        maininfileoutfileoption
*        infileInputfilename.
*        outfileOutputfilename.
*        option-cCompressdatafile.
*                -eExpanddatafile.
*/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include"bitio.h"
#include"main.h"

voidusage_exit();
voidprint_ratios();
longfile_size();

intmain(argc,argv)
```

ลิสติง 2 (ต่อ)

```
intargc;
char*argv[];
{
setbuf(stdout,NULL);
if(argc<3)
usage_exit(argv[0]);

if(strcmp(argv[3],"-c")==0){
BIT_FILE*output;
FILE*input;

input=fopen(argv[1],"rb");
if(input==NULL)
 fatal_error("Erroropening%sforinput\n",argv[1]);
output=OpenOutputBitFile(argv[2]);
if(output==NULL)
fatal_error("Erroropening%sforoutput\n",argv[2]);
printf("\nCompressing%sto%s\n",argv[1],argv[2]);
printf("Using%s\n",CompressionName);
CompressFile(input,output);
CloseOutputBitFile(output);
fclose(input);
print_ratios(argv[1],argv[2]);
return(0);
}elseif(strcmp(argv[3],"-e")==0){
FILE*output;
BIT_FILE*input;

input=OpenInputBitFile(argv[1]);
if(input==NULL)
        fatal_error("Erroropening%sforinput\n",argv[1]);
output=fopen(argv[2],"wb");
if(output==NULL)
        fatal_error("Erroropening%sforoutput\n",argv[2]);
printf("\nExpanding%sto%s\n",argv[1],argv[2]);
printf("Using%s\n",CompressionName);
ExpandFile(input,output);
CloseInputBitFile(input);
fclose(output);
```

ลิสติง 2 (ต่อ)

```
putc('\n',stdout);
return(0);
}else
usage_exit(argv[0]);
}

voidusage_exit(prog_name)
char*prog_name;
{
char*short_name;
char*extension;

short_name=strrchr(prog_name,'\\');
if(short_name==NULL)
short_name=strrchr(prog_name,'/');
if(short_name==NULL)
short_name=strrchr(prog_name,':');
if(short_name!=NULL)
short_name++;
else
short_name=prog_name;
extension=strrchr(short_name,'.');
if(extension!=NULL)
*extension='\0';
printf("\nUsage:%s%s\n",short_name,Usage);
exit(0);
}

#ifndefSEEK_END
#defineSEEK_END2
#endif

longfile_size(name)
char*name;
{
longeof_ftell;
FILE*file;

file=fopen(name,"r");
if(file==NULL)
return(0L);
fseek(file,0L,SEEK_END);
eof_ftell=ftell(file);
fclose(file);
return(eof_ftell);
}

voidprint_ratios(input,output)
char*input;
```

ลิสติง 2 (ต่อ)

```
char*output;
{
longinput_size;
longoutput_size;
intratio;

input_size=file_size(input);
if(input_size==0)
input_size=1;
output_size=file_size(output);
ratio=100-(int)(output_size*100L/input_size);
printf("\nInputbytes:%ld\n",input_size);
printf("Outputbytes:%ld\n",output_size);
if(output_size==0)
output_size=1;
printf("Compressionratio:%d%%\n",ratio);
}
```

ลิสติง 3 HUFFMAN.C

```
/*
*HUFF.CThisistheHuffmancodingmodule.
*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include"bitio.h"
#include"main.h"

/*
*TheNODEstructureisanodeintheHuffmandecodingtree.Ithasa
*count,whichisitsweightinthetree,andthenodenumbersofits
*twochildren.Itjustholdstheintialcountforeachofthesymbols,
*sincethecountmemberiscontinuallybeingmodifiedasthetreegrows.
*/
typedefstructtree_node{
unsignedintcount;
intchild_0;
intchild_1;
}NODE;

/*
*AHuffmantreeissetupfordecoding,notencoding.Whenencoding,
*Ifirstwalkthroughthetreeandbuildupatableofcodesfor
*eachsymbol.ThecodesarestoredinthisCODEstructure.
```

ลิสติง 3 (ต่อ)

```
*/
typedefstructcode{
unsignedintcode;
intcode_bits;
}CODE;

/*
*ThespecialEOSsymbolis256,thefirstavailablesymbolafterall
*ofthepossiblebytes.Whendecoding,readingthissymbols
*indicatesthatallofthedatahasbeenreadin.
*/
#defineEND_OF_STREAM256

voidcount_bytes();
voidscale_counts();
intbuild_tree();
voidconvert_tree_to_code();
voidoutput_counts();
voidinput_counts();
voidcompress_data();
voidexpand_data();

/*
*ThesetwostringsareusedbyMAIN-C.CandMAIN-E.Ctoprint
*messagesofimportancetotheuseroftheprogram.
*/
char*CompressionName="staticorder0modelwithHuffmancoding";
char*Usage="infileoutfile[-c/-e]\n\t-cCompression\n\t-eExpansion\n";

voidCompressFile(input,output)
FILE*input;
BIT_FILE*output;
{
unsignedlong*counts;
NODE*nodes;
CODE*codes;
introot_node;

counts=(unsignedlong*)calloc(256,sizeof(unsignedlong));
if(counts==NULL)
fatal_error("Errorallocatingcountsarray\n");
if((nodes=(NODE*)calloc(514,sizeof(NODE)))==NULL)
        fatal_error("Errorallocatingnodesarray\n");
if((codes=(CODE*)calloc(257,sizeof(CODE)))==NULL)
        fatal_error("Errorallocatingcodesarray\n");
count_bytes(input,counts);
scale_counts(counts,nodes);
output_counts(output,nodes);
root_node=build_tree(nodes);
```

ลิสติง 3 (ต่อ)

```
convert_tree_to_code(nodes,codes,0,0,root_node);
compress_data(input,output,codes);
free((char*)counts);
free((char*)nodes);
free((char*)codes);
}
voidExpandFile(input,output)
BIT_FILE*input;
FILE*output;
{
NODE*nodes;
introot_node;

if((nodes=(NODE*)calloc(514,sizeof(NODE)))==NULL)
fatal_error("Errorallocatingnodesarray\n");
input_counts(input,nodes);
root_node=build_tree(nodes);
expand_data(input,output,nodes,root_node);
free((char*)nodes);
}

voidoutput_counts(output,nodes)
BIT_FILE*output;
NODE*nodes;
{
intfirst;
intlast;
intnext;
inti;

first=0;
while(first<255&&nodes[first].count==0)
        first++;

for(;first<256;first=next){
        last=first+1;
        for(;;){
        for(;last<256;last++)
                if(nodes[last].count==0)
                break;
        last--;
        for(next=last+1;next<256;next++)
                if(nodes[next].count!=0)
                break;
        if(next>255)
                break;
        if((next-last)>3)
                break;
        last=next;
```

ลิสติง 3 (ต่อ)

```
        };
        if(putc(first,output->file)!=first)
        fatal_error("Errorwritingbytecounts\n");
        if(putc(last,output->file)!=last)
        fatal_error("Errorwritingbytecounts\n");
        for(i=first;i<=last;i++){
if(putc(nodes[i].count,output->file)!=
(int)nodes[i].count)
                fatal_error("Errorwritingbytecounts\n");
        }
}
if(putc(0,output->file)!=0)
        fatal_error("Errorwritingbytecounts\n");
}

voidinput_counts(input,nodes)
BIT_FILE*input;
NODE*nodes;
{
intfirst,last;
inti,c;

for(i=0;i<256;i++)
        nodes[i].count=0;
if((first=getc(input->file))==EOF)
        fatal_error("Errorreadingbytecounts\n");
if((last=getc(input->file))==EOF)
        fatal_error("Errorreadingbytecounts\n");
for(;;){
        for(i=first;i<=last;i++)
        if((c=getc(input->file))==EOF)
                fatal_error("Errorreadingbytecounts\n");
        else
                nodes[i].count=(unsignedint)c;
        if((first=getc(input->file))==EOF)
        fatal_error("Errorreadingbytecounts\n");
        if(first==0)
        break;
        if((last=getc(input->file))==EOF)
        fatal_error("Errorreadingbytecounts\n");
}
nodes[END_OF_STREAM].count=1;
}

#ifndefSEEK_SET
#defineSEEK_SET0
#endif
```

ลิสติง 3 (ต่อ)

```
voidcount_bytes(input,counts)
FILE*input;
unsignedlong*counts;
{
longinput_marker;
intc;

input_marker=ftell(input);
while((c=getc(input))!=EOF)
        counts[c]++;
fseek(input,input_marker,SEEK_SET);
}

voidscale_counts(counts,nodes)
unsignedlong*counts;
NODE*nodes;
{
unsignedlongmax_count;
inti;

max_count=0;
for(i=0;i<256;i++)
if(counts[i]>max_count)
        max_count=counts[i];
if(max_count==0){
        counts[0]=1;
        max_count=1;
}
max_count=max_count/255;
max_count=max_count+1;
for(i=0;i<256;i++){
        nodes[i].count=(unsignedint)(counts[i]/max_count);
        if(nodes[i].count==0&&counts[i]!=0)
        nodes[i].count=1;
}
nodes[END_OF_STREAM].count=1;
}

intbuild_tree(nodes)
NODE*nodes;
{
intnext_free;
inti;
intmin_1,min_2;

nodes[513].count=0xffff;
for(next_free=END_OF_STREAM+1;;next_free++){
        min_1=513;
```

ลิสติง 8 (ต่อ)

```
        min_2=513;
        for(i=0;i<next_free;i++)
if(nodes[i].count!=0){
if(nodes[i].count<nodes[min_1].count){
min_2=min_1;
min_1=i;
}elseif(nodes[i].count<nodes[min_2].count)
min_2=i;
}
        if(min_2==513)
        break;
        nodes[next_free].count=nodes[min_1].count
        +nodes[min_2].count;
        nodes[min_1].count=0;
        nodes[min_2].count=0;
        nodes[next_free].child_0=min_1;
        nodes[next_free].child_1=min_2;
}
next_free--;
return(next_free);
}

voidconvert_tree_to_code(nodes,codes,code_so_far,bits,node)
NODE*nodes;
CODE*codes;
unsignedintcode_so_far;
intbits,node;
{
if(node<=END_OF_STREAM){
        codes[node].code=code_so_far;
        codes[node].code_bits=bits;
        return;
}
code_so_far<<=1;
bits++;
convert_tree_to_code(nodes,codes,code_so_far,bits,nodes[node].child_0);
convert_tree_to_code(nodes,codes,code_so_far|1,bits,nodes[node].child_1);
}

voidcompress_data(input,output,codes)
FILE*input;
BIT_FILE*output;
CODE*codes;
{
intc;

while((c=getc(input))!=EOF)
        OutputBits(output,(unsignedlong)codes[c].code,codes[c].code_bits);
OutputBits(output,(unsignedlong)codes[END_OF_STREAM].code,codes[END
_OF_STREAM].code_bits);
```

ลิสติง 8 (ต่อ)

```
}

voidexpand_data(input,output,nodes,root_node)
BIT_FILE*input;
FILE*output;
NODE*nodes;
introot_node;
{
intnode;

for(;;){
node=root_node;
do{
if(InputBit(input))
node=nodes[node].child_1;
else
node=nodes[node].child_0;
}while(node>END_OF_STREAM);
        if(node==END_OF_STREAM)
break;
if((putc(node,output))!=node)
fatal_error("ErrortryingtowriteexpandedbytetooutputI");
}
```